10-23-2013

Editing R2RML Mappings Made Easy

Kunal Sengupta
sengupta.4@wright.edu

Peter Haase

Michael Schmidt

Pascal Hitzler
pascal.hitzler@wright.edu

Editing R2RML Mappings Made Easy.

Kunal Sengupta[1,2⋆], Peter Haase[1], Michael Schmidt[1], and Pascal Hitzler[2]

[1] fluid Operations AG, Walldorf, Germany
[2] Wright State University, Dayton OH 45435, USA

Abstract. The new W3C standard R2RML[3] defines a language for expressing mappings from relational databases to RDF, allowing applications built on top of the W3C Semantic Technology stack to seamlessly integrate relational data. A major obstacle in using R2RML, though, is the creation and maintenance of mappings. In this demo, we present a novel R2RML mapping editor which provides a user interface to create and edit mappings interactively even for non-experts.

1 Introduction

The RDB to RDF mapping language (R2RML) has recently become a W3C standard for creating mappings from relational databases to RDF. This enables many semantic web applications to integrate easily with relational databases. Although very useful, we observe certain problems with the adoption of the new standard: (1) creating R2RML rules manually is a time consuming process, (2) even simple rules could be syntactically heavy in terms of usage of the R2RML vocabulary, and (3) a steep learning curve is involved in gaining expertise of this new language. With these issues in mind, we have developed an R2RML mapping editor that provides an intuitive interface to the users to create, edit and manage R2RML mappings. The editor is fully integrated into the Information Workbench [1], an Open Source platform for Linked Data application development.

2 R2RML by Example

A mapping rule in this language is referred to as a TriplesMap. Example 1 illustrates an R2RML mapping rule in Turtle[4] syntax. The R2RML mapping in this example has three components: The predicate `rr:logicalTable` points to a structure selecting a table, view, or SQL query from the relational database that is mapped into RDF; in this example, a custom SQL query is used to retrieve the columns id, gid and length from the table recording. The predicate `rr:subjectMap` points to a structure defining how rows from the logical table query results are transformed into possibly typed, subjects; in the example, we create subjects by instantiating the specified `rr:template` with the values from

⋆ Work performed while at fluid Operations.
[3] See: http://www.w3.org/TR/r2rml/
[4] See `http://www.w3.org/TR/turtle/`.

the `{gid}` column, and specify `mo:recording` as the type of these subjects. The predicate `rr:predicateObjectMap` points to a structure defining property-value pairs for the subjects defined by the `rr:subjectMap`; in the example, for every subject we create a property `mo:duration`, pointing to a literal created by column `length` of the source database.

Example 1.

```
rr:logicalTable
  [rr:sqlQuery
     """SELECT id,  gid, length FROM  musicbrainz.recording
     WHERE musicbrainz.recording.length IS NOT NULL"""]
rr:subjectMap
  [rr:class mo:recording ;
   rr:template "http://musicbrainz.org/artist/{gid}\#\_"]
rr:predicateObjectMap
   [rr:objectMap
     [ rr:column "length" ;
       rr:datatype <http://www.w3.org/2001/XMLSchema\#float> ] ;
     rr:predicate  mo:duration].
```

As can be seen from the example, the syntax to denote the mapping is rather convoluted and requires deep technical knowledge of R2RML, its language constructs, syntactic details (such as quotation), and the R2RML vocabulary itself.

3 The Mapping Editor

The editor that we have developed in order to overcome these technicalities features an intuitive user interface which hides the R2RML vocabulary details from the user, provides access to relational metadata of the relational database for which the mappings are to be created, provides instant feedback where, at each step, the user has the option to preview the triples that would result from the mapping that is being created, and supports most of R2RML vocabulary except `rr:graph`, which we plan to include as we continuously evolve the editor. In the following we briefly describe the various steps of the editor's wizard workflow.

Step 1 **Datasource, Base URI Selection.** The first step in the wizard lets the user choose the datasource for which mappings shall be defined, i.e. by design the editor supports the maintenance of mappings over multiple relational data sources. As part of the datasource selection, the user may also choose a base URI for URI construction templates specified in the upcoming steps.

Step 2 **R2RML Rule Selection.** After datasource selection, the editor loads and displays previously saved mappings for the datasource. At this point the user may choose to edit an existing rule or add a new one. The editor also provides a filter using which the mappings could be looked up based on involved table names, concept names and property names.

Step 3 **Logical Table Selection.** In this step, the editor provides various options to choose a logical table. The logical table could be a database table, view or a custom SQL query written by the user. If the option to add a database table is selected, all the tables from the database schema

are displayed. Options to preview the table's data and metadata are also available. On the other hand, if the option of SQL query is chosen then an SQL editor is shown where the user can type SQL queries and preview the results.

Step 4 **Subject Map Creation.** Once a logical table has been selected, the editor guides the user to a subject template creation step. By the click of a button, the user can carry over columns that should be used to generate the template. Additionally, classes (i.e., types) can be added to the subject by using the auto-complete field rdf:type, which displays all the classes that are present in the system. The editor is flexible enough such that multiple classes can be added. At this stage a preview of the generated triples can be accessed as well. This provides feedback to the user such that they can adjust the values in order to obtain the intended results.

Step 5 **Predicate-Object Maps Creation.** After the subject map creation, predicate-object maps can be added to the mapping. Again, the editor provides flexibility for adding as many predicate-object pairs as needed. Additionally, for object maps all types of terms defined in R2RML are supported, namely constant terms, column values, templates, and object references. The object reference option lets the user choose from the previously defined mappings. Furthermore, on choosing the appropriate rule the child and parent columns are prepopulated from the corresponding logical tables. Again, multiple join conditions can be added through the interface. The preview triples option is also available at this stage, again providing the user with valuable feedback. Fig. 1 depicts the screen corresponding to this step.

Step 6 **Textual Representation.** Finally, an intuitive textual representation of the mapping rule is displayed. At this point the user could go back to any previous step to modify the mapping or otherwise save the mapping.

The demo video: `https://www.youtube.com/watch?v=PfP2wmWw9-o` is available online, the video includes a very brief introduction to R2RML and motivation for an editor to create R2RML mappings. The remainder of the demonstration shows a use case of mapping MusicBrainz[5] database tables to the Music ontology.[6] In particular, we demonstrate two example mappings, where each step of the mapping creation process is explained and special features of the editor are highlighted.

Although tools like Karma [2], Ontop [3], Topbraid Composer[7], and Neon Toolkit[8] can be used for mapping relational and other datasources to RDF, the editor presented here provides the following advantages (1) It is the first editor to fully support R2RML mappings, (2) The preview feature is unique as the users get instant feedback and can modify the mappings iteratively, (3)

[5] MusicBrainz: https://wiki.musicbrainz.org/Next_Generation_Schema
[6] Music ontology: http://musicontology.com
[7] See `http://www.topquadrant.com/products/TB_Composer.html`
[8] See `http://neon-toolkit.org/wiki/Main_Page`

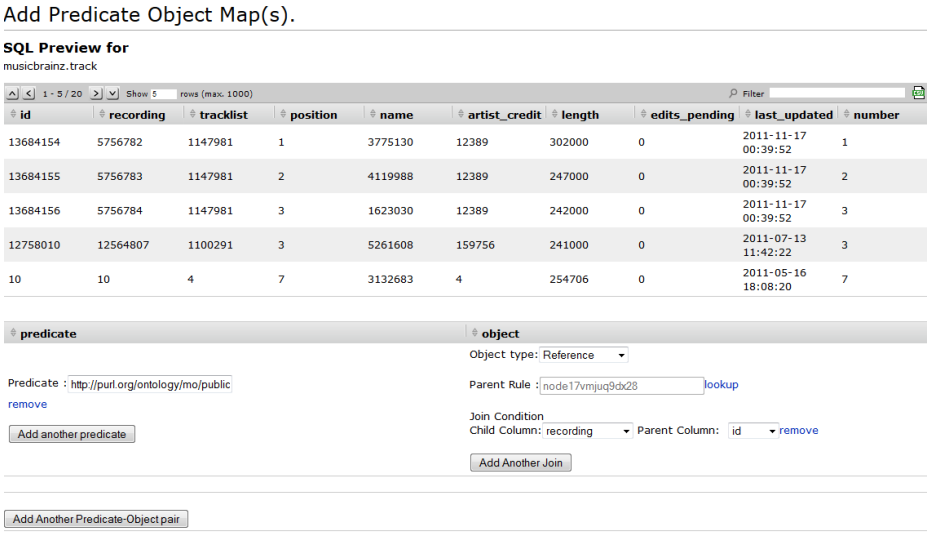

Fig. 1. Screen for adding predicate-object maps

It provides easy management of R2RML mappings, (4) It provides a search utility, where users can find mappings based on table names, concept names and property names. Apart from these advantages, we are going to further improve the editor by adding features like automatic mapping suggestions and on click default mapping [4] generation.

References

1. Hartig, O., Harth, A., Sequeda, J. (eds.): Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23, 2011, CEUR Workshop Proceedings, vol. 782. CEUR-WS.org (2011)
2. Knoblock, C.A., Szekely, P.A., Ambite, J.L., Goel, A., Gupta, S., Lerman, K., Muslea, M., Taheriyan, M., Mallick, P.: Semi-automatically mapping structured sources into the semantic web. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, Ó., Presutti, V. (eds.) The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7295, pp. 375–390. Springer (2012)
3. Rodriguez-Muro, M., Calvanese, D.: -ontop- framework (2012), `http://obda.inf.unibz.it/protege-plugin/`
4. Sequeda, J., Arenas, M., Miranker, D.P.: On directly mapping relational databases to RDF and OWL. In: Mille, A., Gandon, F.L., Misselis, J., Rabinovich, M., Staab, S. (eds.) Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012. pp. 649–658. ACM (2012)